Computational Methods for Radiance

Render “the full variety offered by the direct observation of objects.”
Methods for Plenoptic 1.0
Computing with Radiance

- Goal: Render “the full variety offered by the direct observation of objects.”

- Computational tasks:
  - Interpreting a digital plenoptic image as radiance
  - Rendering radiance to image
  - Algorithms for transforming radiance
  - Real-time interactive implementation using GPUs
Radiance Representation (Plenoptic 1.0)

- Sensor image represents sampled radiance
  - Position is sampled by microlenses “as pixels”
Radiance Representation (Plenoptic 1.0)

Captured by Sensor

Interpreted as Radiance
Radiance Representation (Plenoptic 1.0)
Radiance Representation (Plenoptic 1.0)

- Plenoptic image is a “flat” 2D array of 2D arrays
- 4D array
- “Position major”
Radiance Representation (Plenoptic 1.0)
Radiance Representation (Plenoptic 1.0)

- Plenoptic image is a “flat” 2D array of 2D arrays
- 4D array
- “Direction major”
Camera Arrays

- The most popular lightfield camera is simply an array of conventional cameras, like the Stanford array.

- Alternatively, an array of lenses/prisms with a common sensor, like the Adobe array.
A traditional image (a picture) is formed by integrating rays from every direction at each pixel.

\[ I(q) = \int_P r(q, p) \, dp \]
Rendering the Wide Variety

- Averaging recovers traditional picture
- Wide variety can also be rendered
  - Different aperture
  - Different viewpoint
  - Different focus
  - Different depth of field
  - Stereo
  - High dynamic range
  - Super resolution
  - ...
Different Aperture

- A smaller aperture is a smaller set of directions
Different Viewpoints

- Different viewpoint is different direction
- Render different directions (or sets of directions)
Different Viewpoints

- Different viewpoint is different direction
- Render different directions (or sets of directions)
Different Viewpoints

- Different viewpoint is different direction
- Render different directions (or sets of directions)
Example [Ren Ng]
Example [Ren Ng]
Example [Ren Ng]
Refocusing

- When we refocus a camera, we change the distance from the lens to the sensor.
- Same object is no longer in focus.
Computational Refocusing

- Change the distance (translate) computationally
- Two different radiances, $r_1$ and $r_2$
Computational Refocusing

- We capture radiance $r_1$. How can we compute $r_2$?
- We need translation transform of the radiance.
Algorithm: Computational Refocusing

- Apply shearing transformation: $r'(q, p) = r(q - tp, p)$

Then render the new image: $I(q) = \int r(q, p) dp$
Computational Refocusing (Ren Ng)
Computational Refocusing (Ren Ng)
Methods for Plenoptic 2.0
Radiance Representation (Plenoptic 2.0)

- Sensor image samples the radiance
  - Each microlens image samples in position and direction
Radiance Representation (Plenoptic 2.0)

Captured by Sensor

Interpreted as Radiance

Focused plenoptic camera
Radiance Representation (Plenoptic 2.0)
Radiance Representation (Plenoptic 2.0)

- Plenoptic 2.0 image is a “flat” 2D array of 2D arrays
- Radiance is 4D array
- “Direction major” (approximately, in the sense of tilting)
Rendering Full Aperture Image from 2.0 Data

\[ I[i, j] = \frac{1}{N^2} \sum_{m,n} r[m, n, i, j] \]
Rendering One View from 2.0 Data

\[ I[i, j] = r[m, n, i, j] \]
Plenoptic 2.0 Rendering

- Microlens Image
- Captured Radiance
- Rendered Image
- Full Resolution Rendering

Diagram showing the process from captured radiance to rendered image through microlens images and patch calculations.
Plenoptic 2.0 Rendering Example
Plenoptic 2.0 Rendering Example
Rendering the Wide Variety

- Averaging recovers traditional picture
- Wide variety can also be rendered
  - Different aperture
  - Different viewpoint
  - Different focus
  - Different depth of field
  - Stereo
  - High dynamic range
  - Super resolution
  - ...
Plenoptic 2.0 Rendering Parallax
Plenoptic 2.0 Rendering Parallax

Captured Radiance

Microlens Image

Patch

$P$

$P \cdot N_y$

$P \cdot N_x$

Rendered Image

Full Resolution Rendering
Plenoptic 2.0 Rendering Parallax
Plenoptic 2.0 Rendering Parallax
Plenoptic 2.0 Rendering Parallax
Plenoptic 2.0 Rendering Parallax
Plenoptic 2.0 Refocusing
Plenoptic 2.0 Refocusing
Plenoptic 2.0 Refocusing
Plenoptic 2.0 Refocusing
Efficient Implementation with GPU

Real-Time Radiance Rendering and Transforms
Graphics Processing Units

- Radiance processing is computationally expensive
- CPU clock speeds stalled at 3.0GHz
- Nvidia GTX 295:
  - 1.8 Tflop
  - $500
GPU Programming

- Basic alternatives for programming GPU: General purpose or graphics-based
- Open GL Shader Language (GLSL) a natural fit
Rendering with GPU using Open GL

- Read in plenoptic radiance image
- Create 2D texture object for radiance
- Serialize image data to Open GL compatible format
- Define the texture to OpenGL

```python
image = Image.open("lightfield.png")
str_image = image.tostring("raw", "RGBX", 0, -1)

glActiveTexture(GL_TEXTURE0)
lfTexture = glGenTextures(1)
glBindTexture(GL_TEXTURE_RECTANGLE_ARB, lfTexture)
glTexImage2D(GL_TEXTURE_RECTANGLE_ARB, 0, 3,
             image.size[0], image.size[1], 0,
             GL_RGBA, GL_UNSIGNED_BYTE, str_image)
```
GLSL Implementation of Rendering
GLSL Implementation of Rendering

- **Captured Radiance**
- **Microlens Image**
- **Patch**
- **Rendered Image**

Flowchart:
- \( N_x \) and \( N_y \) directions
- \( P \) and \( P \cdot N_x \) directions
- \( n_x \) and \( n_y \) directions

**Full Resolution Rendering**
GLSL Implementation of Rendering

```glsl
#extension GL_ARB_texture_rectangle : enable

uniform sampler2DRect flat;
uniform float mu;
uniform float M;

greater main()
{
    vec2 p = floor(gl_TexCoord[0].st / mu);
    vec2 qp = (gl_TexCoord[0].st / blockSize - p) * M + 0.5*(mu - M);
    vec2 fx = p * mu + qp;

    gl_FragColor = texture2DRect(flat, fx);
}
```
GLSL Implementation (Live Demo)

- Rendering
- Parallax
- Refocusing